The aim of this lab session is to introduce you to software-defined radio (SDR). At the end of the lab, you should be able to:

- Describe the basic functionality of SDR
- Learn about the development tool GNU Radio
- Construct basic block diagrams in GNU Radio to communicate with the SDR

# 1 Overview of Software Defined Radio

Software-defined radios (SDRs) are slowly emerging as important elements of communication systems as they greatly help us in understanding the theoretical concepts of wireless communications using practical exercises. A complete standalone communication system can be constructed with the help of SDRs that includes the transmission and reception of electromagnetic waves. Although the term, SDR may indicate that there is extensive knowledge of software development required, in reality, we rely on a lot of readily available tools in the form of blocks through which we can achieve the desired functionality. We must be cognizant of the fact of the use of SDRs involves working with the radio wave spectrum which is regulated by national agencies in every country.

Decades ago, SDRs were mainly used in military applications where there is often a need to construct on-demand communication networks. With the advancements in hardware design, these days the cost of SDRs has become affordable and therefore, we can now develop our own flexible communication networks. The core component of a modern SDR is the analogue-to-digital converter (ADC) after which digital signal processing (DSP) can be implemented in software. The DSP based implementation allows constructing of complex components that would otherwise be very complicated using analogue or hardware means. The block-level architecture of an SDR is shown in Fig. 1. The radio frequency (RF) frontend mainly consists of flexible hardware components that can be used for both transmission and reception of radio waves. When the SDR is acting as a receiver, the next step is the ADC module that is followed by a digital down-converter (DDC) that converts the intermediate frequency signal to baseband. This step allows convenience in software processing in the DSP module which is fully programmable. Processes such as modulation, signal processing, encoding, and equalisation can be performed in the DSP module. When the SDR acts as a transmitter, the baseband signal is upconverted using a digital

up-converter (DUC) followed by a digital-analogue converter (DAC) before the RF frontend transmits the signal as a radio wave.

There are two types of SDRs; the first one RTL-SDR are very low-cost, but we can only receive the radio waves. They can be easily connected to a personal computer via the universal serial bus (USB) interface once the appropriate drivers have been installed. The second type, which we will be using in this lab, acts as a high-performance transceiver SDR. The communication is also enabled via the USB interface, however, there are special drivers (`libiio`) required for proper working.

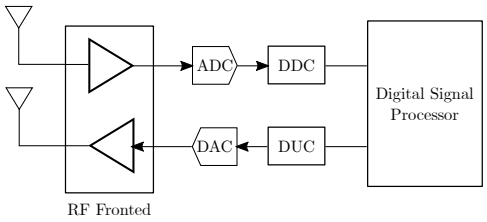


Figure 1: Block diagram of a typical SDR transceiver.

### 1.1 ADALM-PLUTO

One of the well-known high-performance SDRs available today is the ADALM-PLUTO having a frequency range of 0.3 GHz to 3.8 GHz. The connectivity with the host computer is provided by the USB 2 interface enabled by `libiio` and remote network driver interface specification (RNDIS). Due to this, we can simply access the Pluto SDR using the IP address, `192.168.2.1`. The Pluto DSP architecture is based on the Xilinx Zen 7010 field-programmable grid array (FPGA) in which we can store and run different functionalities. The RF frontend is based on the AD9361 transceiver that consists of modules such as a local oscillator (LO) that can be tuned, RF filters and GSM antennas that have a radiation pattern similar to a dipole antenna. Further details about the Pluto SDR can be found here.

## 2 Initial Configuration

### 2.1 Install Pluto USB Drivers

Before we start using the Pluto SDR, it is important to install relevant drivers. It is highly recommended that you use a Linux based machine before proceeding [1]. Although there are ways to install a Ubuntu (Linux

[1] Latest updates of macOS (ver. $\geq$ 10.15) now prevent the installation of the RNDIS interface due to which communication between the computer and SDR module. Please note that the Microsoft Windows 10 can also do the job, albeit with some additional steps.

flavour) virtual machine on your host machines, interfacing with the actual hardware may create potential issues. Anyway, instructions for installation can be found for Windows here, and macOS here here. Please note that this requires the installation of an additional software called `Virtual Box`. The drivers for Linux and Windows operating systems can be found here:

- Linux Driver
- Windows Driver - latest version 0.7

## 2.2 Install LIBIIO Drivers

In order to interface embedded devices with a host computer, a `C/C++` library that enables generic access to industrial input output (IIO) devices has been developed that includes ADCs and DACs. Rather than installing the drivers using a setup file, here we will build the drivers using some `Linux` terminal commands. Try with the following commands in the `Linux` terminal. Type the commands after the symbol `$` one by one in the terminal window:

```
    $ sudo apt-get install libxml2 libxml2-dev bison flex libcdk5-dev cmake
    $ sudo apt-get install libaio-dev libusb-1.0-0-dev      libserialport-dev libxml2-dev
↪   libavahi-client-dev      doxygen graphviz
    $ git clone https://github.com/pcercuei/libini.git
    $ cd libini
    $ mkdir build && cd build && cmake ../ && make && sudo make install
    $ cd
    $ git clone https://github.com/analogdevicesinc/libiio.git
    $ cd libiio
    $ cmake ./
    $ make all
    $ sudo make install
    $ PATH=/usr/lib/:$PATH
```

The first two lines above install some required packages before we can build `libiio`. We then download the package contents of `libini` using the `git` command and then use `make` commands to build and install the package. The package `libiio` is installed in a similar manner. The last line above ensures that the `libiio`directory can be found when other packages are looking for it. Detailed instructions can be found at https://wiki.analog.com/resources/tools-software/linux-software/libiio.

On Windows, the latest installation file can be downloaded from here.

Once `libiio` is installed, we need to make sure, everything is running fine. One indication is that the `LED1` on the ADALM-Pluto SDR is blinking when we plug in the module to the computer. As shown in Fig. 2, the `Ready`


Figure 3: A picture of Analog Devices ADALM-Pluto SDR to be used in the lab.

LED should also be turned ON. To further validate everything is running as expected, type in the Ubuntu terminal (command prompt in Windows) the following:

```
$ iio_info -s
```

You should receive information about the connected Pluto SDR such as the serial number, library version of `libiio` and the USB information.

The next step of the configuration is the installation of the visualisation tool `iio-oscilloscope` with the help of which we can plot the data received from the SDRs. The installation process on Linux is similar to what we did earlier:

```
$ apt-get -y install libglib2.0-dev
↪ libgtk2.0-dev libgtkdatabox-dev
↪ libmatio-dev libfftw3-dev libxml2
↪ libxml2-dev bison flex
↪ libavahi-common-dev
↪ libavahi-client-dev -openssl-dev
↪ libjansson-dev cmake libaio-dev
↪ libserialport-dev
$ git clone
↪ https://github.com/analogdevicesinc/libad9361-iio.git
$ cd libad9361-iio
$ cmake ./
$ make
$ sudo make install
$ cd
$ git clone
↪ https://github.com/analogdevicesinc/linux_image_ADI-scripts.git
$ cd linux_image_ADI-scripts
$ chmod +x adi_update_tools.sh
$ sh adi_update_tools
$ cd
$ git clone
↪ https://github.com/analogdevicesinc/iio-oscilloscope.git
$ cd iio-oscilloscope
$ git checkout origin/master
$ mkdir build && cd build
$ cmake ../ && make -j $(nproc)
$ sudo make install
$ export LD_LIBRARY_PATH=./
```

The tool `iio-oscilloscope` can be called from the terminal with the command, `./osc` and a new graphical user interface should appear. Further details on installation and how to use the tool can be found on the website here.


Figure 2: A picture of Analog Devices ADALM-Pluto SDR to be used in the lab.

For Windows, the installers for `labad9361-iio` and `iio-oscilloscope` can be found at here and here respectively. More information on the Windows installation can be found here.

### 2.3 Installing GNU Radio

GNU Radio is an open-source software tool in which we can use block based flowgraphs to implement various functionalities. The hardware can be called either as a source or a sink in the flowgraphs. The installation steps, to be performed in the terminal window are listed below:

```
$ sudo apt install libxml2 libxml2-dev bison flex cmake git libaio-dev libboost-all-dev
$ sudo apt install doxygen
$ sudo apt install libusb-1.0-0-dev
$ sudo apt install libavahi-common-dev libavahi-client-dev
$ sudo apt install bison flex cmake git libgmp-dev
$ sudo apt install liborc-dev
$ sudo apt install gnuradio
```

You should be able to open the GNU Radio by typing `gnuradio-companion` in the terminal window.

The Windows installer for GNU Radio can be downloaded from here.

Before we delve into some exercises, it is important to grasp some concepts of GNU Radio by following the Basics tutorial on the software website.

## 3 Exercise 1 Running an FM Radio Receiver

It is hoped that by now, you will have installed all the drivers and modules required properly. In this exercise, we will use GNU Radio to run a flowgraph through which we can receive radio waves from an FM radio station. To do so, please make sure that the Pluto SDR is connected to the host computer. In order to check whether everything is working fine, open a terminal and type:

```
$ iio_info -s
```

If everything is working properly, you should see the device details as we mentioned earlier. The next step is to download the FM radio GNU Radio source file from Github, that has an extension `.src`. We can use the following command to clone the repo locally to our computer:

```
$ git clone https://github.com/hasantahir/GNU-Radio-Exercises.git
```

which will create a folder `GNU-Radio-Exercises` in the current directory. After opening the source file, you can tune the system by changing the parameter, `fm_station` by double-clicking the parameter block as shown in Fig. 3.

Question - Choose your favourite FM radio station of choice locally from Chengdu.

- Note down the centre frequency of the signal received in GNU radio.
- Take a screenshot of the spectrogram of the received signal.
- Can you find out the bandwidth of the FM signal?

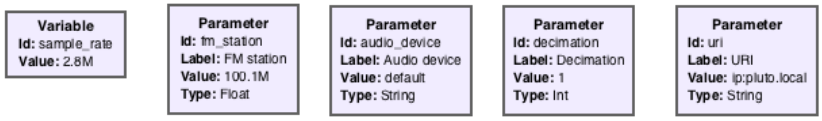


Figure 4: Block showing parameter values in GNU Radio.